ARI Research Note 90-53

Intelligent Support Systems for Hyperknowledge

Gerhard Fischer
University of Colorado

for

Contracting Officer's Representative
Michael Drillings

Basic Research
Michael Kaplan, Director

July 1990

United States Army
Research Institute for the Behavioral and Social Sciences

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

JON W. BLADES
COL, IN
Commanding

Research accomplished under contract for
the Department of the Army

University of Colorado

Technical review by

Michael Drillings

UNCLASSIFIED
URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

. REPORT SECURITY CLASSIFICATION nclassified	1b. RESTRICTIVE MARKINGS --
. SECURITY CLASSIFICATION AUTHORITY -	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
. DECLASSIFICATION/DOWNGRADING SCHEDULE -	
PERFORMING ORGANIZATION REPORT NUMBER(S) -	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARI Research Note 90-53
. NAME OF PERFORMING ORGANIZATION epartment of Computer Science nd Institute of Cognitive cience, University of Colorado 6b. OFFICE SYMBOL (If applicable) --	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Institute
. ADDRESS (City, State, and ZIP Code) niversity of Colorado ampus Box B-19 oulder, CO 80309	7b. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-5600
. NAME OF FUNDING/SPONSORING ORGANIZATION nstitute for the Behavioral nd Social Sciences 8b. OFFICE SYMBOL (If applicable) PERI-BR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-86-C-0143
. ADDRESS (City, State, and ZIP Code) 001 Eisenhower Avenue lexandria, VA 22333-5600	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 61102B PROJECT NO.: 74F TASK NO.: N/A WORK UNIT ACCESSION NO.: N/A

. TITLE (Include Security Classification)
ntelligent Support Systems for Hyperknowledge

. PERSONAL AUTHOR(S)
ischer, Gerhard

a. TYPE OF REPORT nterim	13b. TIME COVERED FROM 86/08 TO 91/08	14. DATE OF REPORT (Year, Month, Day) 1990, July	15. PAGE COUNT 22

. SUPPLEMENTARY NOTATION
ontracting Officer's Representative, Michael Drillings

. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Problem solving; ~~Computer~~ programming; Knowledge representation; Human-computer interaction; Machine learning

. ABSTRACT (Continue on reverse if necessary and identify by block number)

Computer systems can assist in searching, understanding, and creating knowledge in reative problem-solving. We have explored this idea in the context of building a variety f intelligent support systems for high-functionality computer systems, emphasizing the ollowing specific issues: representation of programs as knowledge networks where the code, he documentation, and visual representations are external representations generated from the ame complex internal knowledge structure; user-definable filters to give users control of he parts they would like to see; constraint mechanisms to maintain consistency between nternal and external representations; different browsing systems to explore hyperknowledge paces and design kits as prototypes of hyperknowledge assistants. To increase the usefulness f high-functionality computer systems, they should be constructed as hyperknowledge systems here the intelligent support systems are an integral part of the overall design.

0. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
2a. NAME OF RESPONSIBLE INDIVIDUAL Michael Drillings	22b TELEPHONE (Include Area Code) (202) 274-8722 22c. OFFICE SYMBOL PERI-BR

) Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

INTELLIGENT SUPPORT SYSTEMS FOR HYPERKNOWLEDGE

CONTENTS

LIST OF FIGURES

INTELLIGENT SUPPORT SYSTEMS FOR HYPERKNOWLEDGE

1. Introduction

Modern computer systems are best understood not in their capacity to compute but to serve as *knowledge stores*. And because there is lots of knowledge around, we should not expect that the systems will be small and simple, but they will be large and complex (examples are UNIX systems, LISP machines, etc.; for a quantitative analysis see [Fischer 87a]).

Systems offering such a rich functionality are a mixed blessing: in a very large knowledge store it is much more likely to find something related to what we need, but it is also much more difficult to find something specific. Our empirical investigations indicate that the following problems prevent many users and designers from successfully exploiting the potential of these high-functionality systems: they do not know about the *existence* of tools, how to *access* tools, *when* to use the tools, they do not understand the *results* that tools produce for them, and they cannot combine, adapt, and modify tools to their *specific* needs. A consequence of these problems is that these systems are underused and the broad functionality is of little value. We are strongly convinced that what is needed is *not quantitatively* more information but *qualitatively* new ways to structure and present information.

We claim that in future computer systems, a high percentage of the computational power should be used to make these systems usable, manageable, and useful. Making complex systems useful and usable is not only limited by our inability to produce effective descriptions, but there are inherent limitations in doing this with paper. In the case of program documentation systems, we have argued [Fischer, Schneider 84] that program documentation produced as a separate document by a word processing system has the following disadvantages:

- It is impossible to provide pieces of information automatically.
- It is impossible to maintain consistency between the program and its documentation automatically (or at least semi-automatically).
- It is impossible to generate different external views dynamically from one complex internal structure (e.g., to read a documentation either as a primer or as a reference manual).
- It is impossible to create links between the static description and the dynamic behavior.

Similar arguments apply to *form systems* [Fischer, Rathke 87] where paper representations also have major shortcomings. In forms on paper, *all* cases (e.g., married or not married, employed or unemployed, citizen or foreigner) must be explicitly represented, and paper is incapable of propagating values from one part of a form to another one.

Over the last eight years, we have built systems in the following application domains:

- design environments for high-functionality computer systems [Delisle, Schwartz 86; Fischer, Lemke 87a],
- abstraction hierarchies for specific problem domains to support "human problem-domain communication" [Fischer, Lemke 87b] and reuse and redesign [Fischer 87a; Fischer, Lemke, Rathke 87],
- education (e.g., electronic encyclopedias [Weyer, Borning 85; Weyer 87], help systems [Fischer, Lemke, Schwab 85], and critics [Fischer 87b]),
- personal information systems (e.g., the MEMEX as envisioned by [Bush 45]; the DYNABOOK, serving as the vision for the development of personal computer systems, innovative interfaces, and the SMALLTALK language [Kay 77]; our own effort to develop a conceptual framework and a design for a personal information environment [Fischer, Nieper 87]).

2. Hyperknowledge

Hypertext is in general defined as an approach to information management in which text is stored in a network of nodes connected by links, and this network is meant to be viewed and manipulated interactively. If the nodes do not only contain text, but graphics, audio, video, and other forms of data, the representation form is often called hypermedia. We have chosen to use the term "hyperknowledge" to indicate that the individual nodes can have a rich internal structure and can be linked in a variety of different ways (e.g., our object-oriented knowledge representation formalism OBJTALK [Rathke 86] supports multiple inheritance, constraints, coreference, metaclasses, etc.).[1]

Hyperknowledge (similar to knowledge-based systems) has to address three major problem areas: acquisition of knowledge, representation of knowledge, and utilization of knowledge. There are currently a variety of systems (e.g., the dynamic books constructed by [Weyer 82], the DOCUMENT EXAMINER on the Symbolics LISP machines [Ehrlich, Walker 87]), which only address the third problem: they provide powerful accessing and retrieval methods to a written documentation (produced as a conventional book) to support a variety of access paths to the information store.

2.1 Expectations about Hyperknowledge

Knowledge acquisition, representation, utilization, and dissemination is not independent of the media used in carrying out these objectives. There are various metrics of costs associated using different media. Examples of such metrics dealing with knowledge are the cost of learning, the cost of creating, the cost of comprehension, and the cost of modifying. If a variety of media exist, one has the freedom to choose the one which fits the task best -- eliminating the force to use the same medium for all tasks (hyperknowledge systems are an important step towards the goal that a document or a representation fits the information, not vice versa; hybrid knowledge representation schemes [Stefik et al. 83] try to achieve a similar goal in the area of knowledge representation). As indicated before, paper has severe limitations to represent complex systems. Following we briefly describe a number of problem areas where we expect hyperknowledge representations can make important contributions.

User-centered representations. A user-centered system should support features like: to easily generate personal notes from general documentation, to filter out irrelevant items, to mark important things to remember, and to put the information in a unique perspective, satisfying demands which the designers are unable to anticipate. In order for a system to take the users' needs into account, it has to incorporate information about the users' preferences, skills, and what they have previously looked at. The user should be able to communicate intentions so the system can highlight and prioritize what's important and hide connections and details which are considered irrelevant for the task at hand.

Dynamically generated information. Hyperknowledge architectures should allow information to be generated "on the fly." The visualization tools in our software oscilloscope [Boecker, Fischer, Nieper 86] all share this property; they can take the actual working context into account and visualize the structures the user is working on. This provides a qualitatively different level of support compared to "canned"

[1] Hyperknowledge representations are not too different from semantic nets; one difference is that automatic processing was a more prominent design criteria in the case of semantic nets whereas hyperknowledge spaces should be more supportive towards exploration by humans.

visualizations stored on a video disc. Information on a video disc has to be preselected and cannot take the actual working context into account. The possibility of generating structures dynamically supports an economy of representation by allowing different external representations (e.g., primers or reference manuals) to be generated from the same complex internal knowledge structure, and it makes it it easier to maintain consistency and to update information structures.

Views of reduced complexity. The system should support mechanisms to generate *views of reduced complexity*. Filter mechanisms, where the filters can be defined by the user, allow the generation of views of reduced complexity showing only the information which is relevant for a specific user at a specific time (for an example see section 4.2). If the system is responsible for the generation of the views, knowledge about relevancy must be incorporated (examples are: "fisheye views" [Furnas 86] and the display heuristics used in KAESTLE, a system to visualize list structures [Boecker, Fischer, Nieper 86]).

Support Systems to find information. Powerful features are needed to locate information in a hyperknowledge system. Browsers and navigators (see section 4.1) have turned out to be useful tools for systems organized as inheritance networks. But how do we find out about things in cases where we do not even know that they exist? The retrieval paradigm of incremental query specification [Williams 84] (see section 4.1) augments browsers by helping users with examples to incrementally specify their questions.

Turning systems into coaches and critics. In addition to being just information stores with powerful search mechanisms, hyperknowledge systems should assist the users in carrying out their tasks and solving their problems. Active systems (e.g., acting as critics and consultants [Fischer, Lemke, Schwab 85; Fischer 87b]) are needed which volunteer help in appropriate situations rather than responding to explicit requests. In order to volunteer information, the system needs some understanding of what the user is doing. They should allow users to conjecture, create, and experiment with information instead of just retrieving it (an example being electronic encyclopedias [Weyer, Borning 85]).

Situation models versus system models. A shortcoming of many existing information retrieval systems, like manuals, is that access is by *implementation unit* (e.g., LISP function, UNIX command) rather than by *application goal* on the task level. Current support information is designer-oriented rather than user-oriented. Documentation and help information is structured to describe the system, not to address the problems experienced by the user. We claim that this is the main reason that human assistance, if available on a personal level, is still the most useful source of advice and help. Learners can ask a knowledgeable colleague a question in an infinite variety of ways; they get assistance in formulating the question, and they can articulate their problem in terms of the *situation model* rather than being required to do so in terms of a *system model* [Fischer, Nieper 87]. A knowledge store trying to cover situation models must incorporate user constructs, user-oriented organizations of knowledge, and a presentation component which presents information in the user's concepts and words.

2.2 The Role of Structure

In a recent workshop about personalized intelligent information systems [Fischer, Nieper 87] where many relevant questions about hyperknowledge systems were discussed, the role of structure was identified as a major challenge in system design. The discussion centered around the question "Is structure desirable: yes or no?" Users do not like to be forced to generate structures. In early problem solving stages the

enforcement of structure may get in the way when people start to do something. Different approaches towards this problem are taken by these systems:

- NOTEPAD [Cypher 87] postpones the necessity for creating a structure and requires only a minimal amount of structuring;
- SPATIAL THOUGHT DUMPER [Lewis 87] uses spatial relationships as the only clue;
- NoteCards [Halasz, Moran, Trigg 87; Trigg, Suchman, Halasz 86] requires substantial structuring;
- EUCLID [Smolensky et al. 87] is based on the assumption that argumentation benefits from being structured.

The major design tradeoff is: we would like to take advantage of many structuring principles to retrieve and use information, but we are much less willing to take the overhead into account which is necessary to generate this structure (in Artificial Intelligence systems this questions is often discussed under the heading whether processing should be done at "read-time" or at "question-time"). We don't want to structure *information explicitly, but we do want* to retrieve information using structural properties.

Other questions which were discussed:

- If we impose a structure, which form should it take: hierarchies, inheritance networks, associations, ...?
- Is a structure statically given or generated on demand [Kintsch, Mannes 87]?
- How can effective restructuring be supported?

To be able to cope with large hyperknowledge systems, *structuring principles play an important role* (see for example Simon's ideas about the architecture of complexity [Simon 81]). For many interesting areas, a structure is not given a priori but evolves dynamically. Because little is known at the beginning, there is an almost constant need for *restructuring*. Despite the fact that in many cases users could think of better structures, they stick to inadequate structures, because the effort to change existing structures is too high.

2.3 User Control

The rationale for hyperknowledge systems is to give the user more control. But control does not come for free, and the questions "how much control should users have?" and "how much control do users want to have?" are serious ones. There is no doubt that taking away the control from users in cases where they did not want to have it is regarded as progress in technology (e.g., the automatic transmission in automobiles). Questions which one has to ask are:

- Does end-user modifiability (or customizability) make systems more or less complex? Potentially both possibilities exist: laws are so complex, because they have to anticipate all possible situations and cannot be adjusted dynamically. Forms on papers are complex (with an elaborate branching structure) because they have to cover many different cases. Electronic form systems in which forms are generated dynamically based on users profiles can greatly reduce complexity [Fischer, Rathke 87].
- Should *end-user modifiability* be achieved by *adaptive* (i.e., the system itself changes its behavior based on a model of the user and the task) or *adaptable* systems? Do we have to choose?

- In adaptable systems, how can end-user modifiability be achieved? Is programming necessary for it? Which set of constrained design processes can be supported [Fischer, Lemke 87a] which would eliminate the need for a user to learn details about a system?
- The question to be asked about control should in many cases not be "how much customization is possible?" but rather "how little customization is necessary?"

Using systems like GUIDE[2] or the INFO system for EMACS show that flexibility can be confusing for the producer and the consumer of information. Reading and writing sequential text may put (independent of the fact that we are used to it) a smaller cognitive burden on the user than organizing complex networks.

3. Intelligent Support Systems

Today's software environments require a substantial amount of knowledge to make effective use of their tools and systems. The most common approach in making these systems usable has been to reduce their functionality. Rather than "watering down" the capabilities of these systems, we address with our research the problem of how these systems can be augmented to support their usage. In future computer systems, a high percentage of the computational power should be used to make these systems usable, manageable, and useful. Our system-building efforts are based on the belief that *the intelligence of a complex computer system must contribute to its ease of use.* Truly intelligent and knowledgeable human communicators, such as good teachers, use a substantial part of their knowledge to explain their expertise to others. In the same way, the intelligence of a computer should be applied to providing effective communication.

We need an architecture to support a coherent design strategy that treats intelligent support systems not as add-ons to existing systems but as integral parts of a user-centered design approach. In Figure 3-1 we illustrate a system architecture that we have developed in response to this design criterion.

Over the last few years, we have constructed the following systems belonging to the outer ring:

1. WLISP: a user interface toolkit [Fischer 87a];
2. OBJTALK: an object-oriented, frame-based knowledge representation language [Rathke 86];
3. OBJTALK-BROWSER and OBJTALK-NAVIGATOR: navigational tools which help the user to locate specific items in complex information spaces [Fischer 87a];
4. DOXY: a computer-supported documentation system [Fischer, Schneider 84];
5. PASSIVIST and ACTIVIST: prototypes for passive and active help systems [Fischer, Lemke, Schwab 85];
6. LISP-CRITIC: an example of a critic system [Fischer 87b];
7. KAESTLE: a visualization tool which supports the dynamic generation of graphical representations of LISP data structures [Boecker, Fischer, Nieper 86];
8. WIDES and TRIKIT: two prototypical design kits to assist users in exploiting the functionality of WLISP [Fischer, Lemke 87a; Fischer, Lemke 87b].

A hyperknowledge approach seems to be well suited to support this architecture. In the long run, our perspective of programming as the main activity of creating computer systems will change. Designers

[2]GUIDE is a system manufactured by OWL Inc., Bellevue, WA and is marketed as "hypertext for the Macintosh."

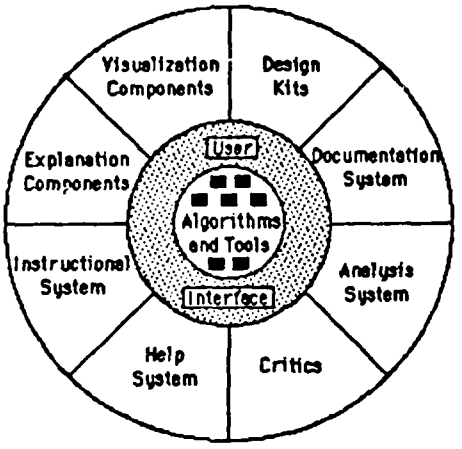

Figure 3-1: An Architecture for Intelligent Support Systems

and users will create knowledge stores, where the executable part (i.e., the code) is just one perspective of it.

4. Description of our Systems

In this section, we will briefly describe three research areas of hyperknowledge in which we have built specific systems:

- support tools to explore hyperknowledge spaces (e.g., browsers, navigators, and incremental query specification);
- representation of programs as knowledge networks where the code (the program listing) is just one external representation, including user-definable filters to see selected parts of a complex internal knowledge structure;
- hyperknowledge assistants which support users in relating the information to their goals and tasks.

4.1 Support Tools to Explore Hyperknowledge Spaces

Over the last years, we have developed OBJTALK [Rathke 86], an object-oriented extension to LISP. OBJTALK has been used to model a number of problem domains (e.g., WLISP, a user interface toolkit [Fischer 87a]) by creating a large number of abstractions and embedding them in a multiple inheritance network (a specific instance of a hyperknowledge space). The OBJTALK-BROWSER and the OBJTALK-NAVIGATOR are two tools to explore this space.

The OBJTALK-BROWSER [Rathke 86] allows the user to "look around" in a system with an unfamiliar structure and search for building blocks (Figure 4-1). It displays the inheritance structure of a system. By selecting a component, its slot descriptions, defaults, triggers, and methods can be analyzed in more detail. Similar tools exist in the programming environment for SMALLTALK [Goldberg 84].

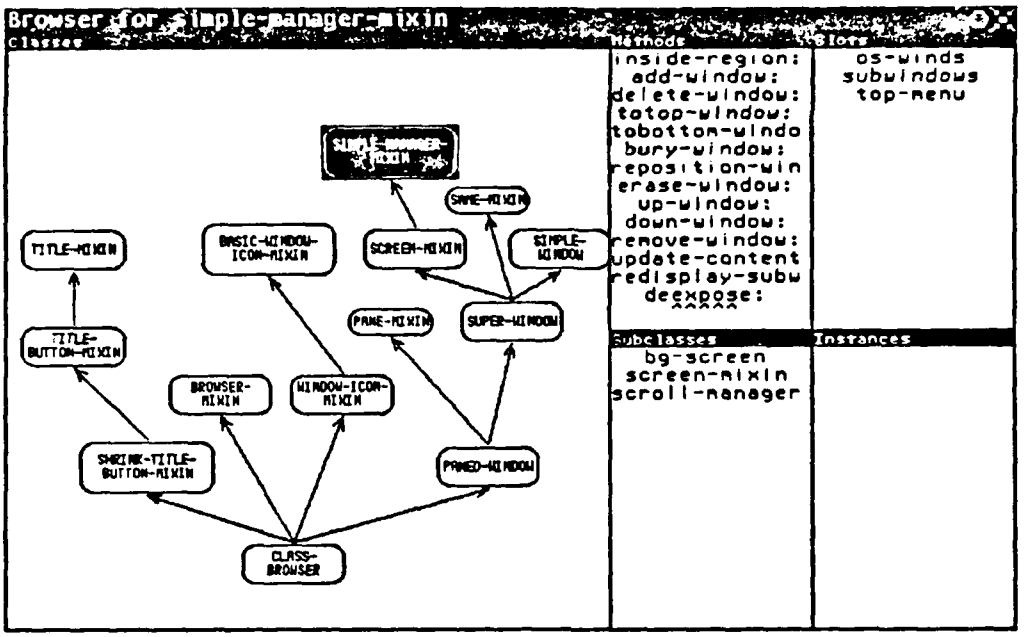

Figure 4-1: The OBJTALK-BROWSER

The OBJTALK-NAVIGATOR. The OBJTALK-BROWSER does not provide any support for locating the inherited slots or methods. Over the last several years working with OBJTALK and WLISP, we have observed that there is a design trade-off in the design of inheritance systems: the goal of reusing building blocks as much as possible leads to a wide distribution of information throughout the inheritance network which can be a major problem in understanding a specific behavior of the system. The OBJTALK-NAVIGATOR is a tool that helps to locate functionality regardless of where it might be defined in the hierarchy.

In addition to the BROWSER, the NAVIGATOR (Figure 4-2) shows all inherited slots and methods, enables the user to selectively view any subset of the inherited slots and methods, and allows the user to go directly to the superclass containing the inherited slot or method without traversing the entire hierarchy. All inherited slots and methods are presented in a scroll menu in the "**Inherited Slots**" and "**Inherited Methods**" windows. The "**Inherited From**" window is initially empty until a slot or method is selected, and then the superclass(es) which contain this slot or method is (are) displayed. In addition the user can specify keywords for searching for all methods or slot names containing this keyword.

It is interesting to note (in the context of *reuse and redesign*) that the NAVIGATOR was built as an extension of the BROWSER, providing *further* empirical evidence for the reuse *and* redesign possibilities offered by our object-oriented approach.

Information Retrieval by Incremental Query Specification and Reformulation. HELGON [Nieper 87] is an information retrieval system based on ARGON [Patel-Schneider, Brachman, Levesque 84] and RABBIT [Williams 84]. The basic principle of these systems is incremental query specification by reformulation. The query is created incrementally, the set of instances matching the current query is shown in one

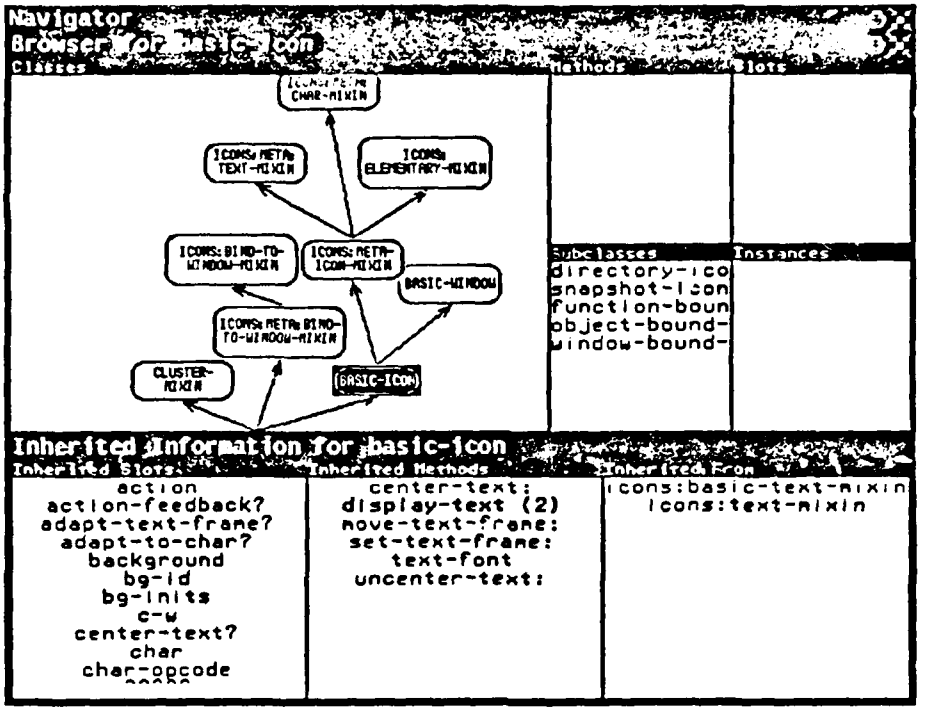

Figure 4-2: The OBJTALK-NAVIGATOR

window, one example instance is shown in another window (Figure 4-3). All information shown can be used to modify the query (e.g., by requiring or prohibiting a certain value, or by selecting an alternative value from a menu). Thus users who are not familiar with the knowledge base (e.g., with relation or attribute names) or who do not know exactly what they want can be guided towards the appropriate information.

A shortcoming of ARGON and RABBIT is that it is not obvious to the users that the frames are organized in a hierarchy, and it is easy to get lost if this hierarchy is not visible. The (currently) major addition of the HELGON system is a graphical display of the frame hierarchy that can be used to add frames to the query.

A major shortcoming of the BROWSER, the NAVIGATOR, RABBIT, and ARGON, which we have partially overcome with HELGON, is that they support only the viewing of information; the user cannot add, change, or delete information at the same interaction level. What we try to achieve with HELGON is to allow operations to add frames or individuals (using existing frames/individuals as templates), to edit existing frames or individuals, to restructure the frame hierarchy (through direct manipulation in the graphical display), and to delete frames or individuals, without forcing the user to learn the underlying knowledge-representation language to update the information network.

In general, it is not possible to show the whole frame hierarchy at once. The amount of information presented to the user can be restricted through filters. The system can start out with the hierarchy displayed to a certain depth only and allow the users to zoom in on parts of interest to them. If information is classified along several orthogonal schemes, one of those schemes can be filtered out, and in

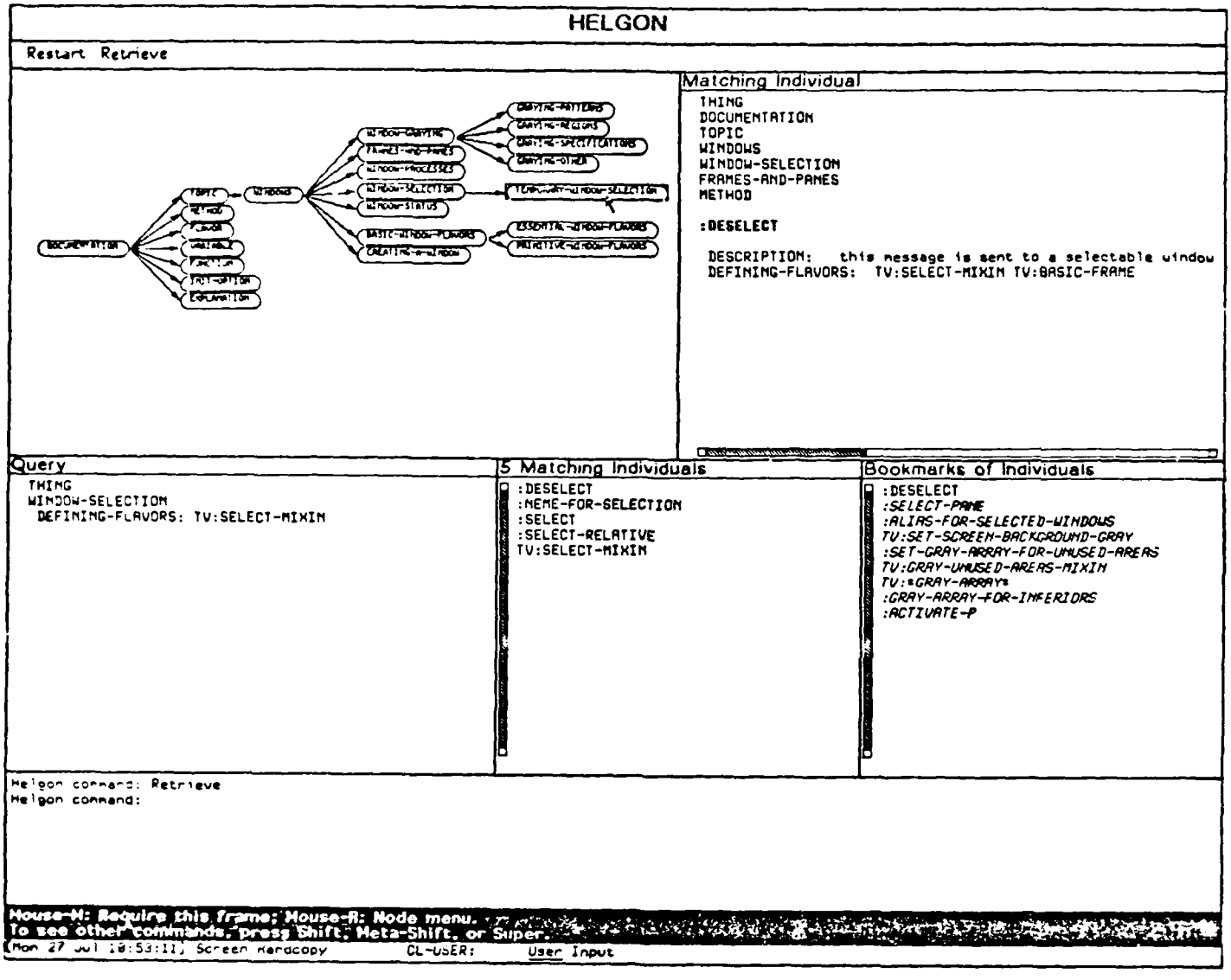

Figure 4-3: HELGON

this way the information can be viewed from different perspectives. Because individuals may be instance of more than one frame and therefore may have slots that are not applicable to all of those frames, slots presented to the user can be filtered depending on the frame that appears in the query.

In an application of the HELGON system, we are modeling the FLAVOR system of the Symbolics LISP machine in an effort to enhance the usability of the existing information and provide alternative access paths in addition to the DOCUMENT EXAMINER (see Figure 4-3).

The DOCUMENT EXAMINER and its Extensions. The Symbolics DOCUMENT EXAMINER [Ehrlich, Walker 87] is the online delivery interface for the complete Symbolics documentation. It is based on an hypertext style information base, and different delivery interfaces provide the capability to generate a set of inter-

esting surface structures. The standard Document Examiner (as offered by Symbolics) tries to retain some of the good aspects of paper sources (e.g., familiarity of the user with books, performance to find information in a book, feeling where you are, skipping around with bookmarks, etc.) with the capabilities of a powerful computing system (e.g., powerful access mechanisms, mouse-sensitive links to other topics).

In our work, we extended the DOCUMENT EXAMINER in two directions. Informal empirical investigations have shown that even after users found some information, there is no guarantee that they are able to understand this information, use it, and modify it to their own needs. *Active examples* [Young 87] augment the DOCUMENT EXAMINER to allow the users to

- execute program code found in the context of the information retrieval process,
- experiment with the example to deepen their understanding,
- explore alternatives and to use them in their own programs.

The second extension augments the DOCUMENT EXAMINER with a RABBIT/ARGON/HELGON style interface to its information base. FLAVARGON uses incremental query specification to support the user in finding a flavor in the huge store of existing flavors (there are more than 2500 flavors in the Symbolics software). Both system components are still in an early stage of development but they have provided us with enough evidence that both of the extensions would make the DOCUMENT EXAMINER a better tool by exploiting unique features of the computer which cannot by offered by a paper-based documentation facility.

4.2 Computer-Supported Documentation Systems

Programs and its associated documentation can be seen as two external views of the same internal structure: the code and other interpretable structures primarily being for the computer and the descriptive elements augmenting the code being directed to the human. If the program and its documentation are both generated from the same source [Knuth 83] then it is much easier to maintain consistency between them. Conventional documentation was designed exclusively to be read by the human, and it was once defined as printed matter that describes or explains how a system of some kind works or should be used.

In the design and implementation of our program documentation system [Fischer, Schneider 84], the knowledge base contained all of the knowledge about a system combined with a set of tools useful for *acquiring, storing, maintaining,* and *using* this knowledge (a sample function description is shown in Figure 4-4). The knowledge base was in part interpreted by the computer to maintain the consistency of the acquired knowledge about structural properties, and it supported the users in debugging and maintaining their programs. Other parts of it were not directly interpretable by the machine which served only as a medium for structured communication between the different users.

Program documentation can be further enhanced by offering *users control* which parts they want to see, serving different groups (clients, designers, programmers, and users) *trying to* perform *different* tasks. The amount and structure of the information offered to these groups of people has to be different, but all external views are generated by using different filters attached to the same knowledge base. Based on the difficulty for the designer to anticipate which view users might be interested in, a fixed set of predesigned filters is too restricted. In our system we have designed a filter kit, where the user can design a specific filter with simple selection processes (Figure 4-5; for details see [Fischer, Schneider 84]).

```
(defobject prem
 (name prem)
 (superclass function-description)
 (status ANALYZED)
 (code
   (def prem (lambda (key1 key2)
        (let ((i (phashit key1 key2)) (a nil))
          (setq a (passociation i key1 key2))
          (cond (a (store ..................)
 (in-package pputget)
 (is-called-by)
 (calls PASSOCIATION PHASHIT)
 (type function)
 (parameter ((key1) (key2)))
 (local-variables (i (TYPE NUMBER)) (a (TYPE NUMBER)))
 (free variables)
 (see-also (pputget-description))
 (history
      ((DEFINED 10/14/1983
          (programmer HDB)
          (reason ""))
       (MODIFIED 12/12/1983
          (programmer HDB)
          (reason "prem didn't work if the property to be deleted
               was the CAR of the appropriate bucket"))))
 (version 2)
 (side-effects (PUTACCESS PUT-GET-HASH-TABLE))
 (purpose "removes properties from the hashtable")
 (description "this function removes the appropriate association-list entry
          from the hashtable. .....))
```

Conventions:
a) bold: slot names of the knowledge structure
b) `typewriter font`: data that can be interpreted, used, and updated by the system
c) normal font: knowledge interpreted, used, and updated by the user
d) CAPITALS: system generated information

Figure 4-4: A Sample Function Description

The program documentation can also be enhanced with the help of the graphical display of data structures generated dynamically with KAESTLE [Boecker, Fischer, Nieper 86].

4.3 Hyperknowledge Assistants

Support for information *retrieval* alone is not good enough -- it gives users no support to relate found information to their problem solving tasks. Our own work has shown that complex construction kits (i.e., sets of building blocks that model a problem domain) by themselves are not good enough to assist the user to solve meaningful problems. To provide additional assistance, we have constructed design kits which go beyond construction kits in that they bring to bear general knowledge about design that is useful

system:
package:
function:
  in-packages
  `callers` ***
    in-packages
    callers
    callees
    purpose
    description
    `code` ***
    see-also
  callees
  `purpose` ***
  description
  `code` ***
  see-also
paper:

Conventions:
a) **bold**: slot names of the knowledge structure
b) `typewriter font` and marked with ***: parts to be shown
c) normal font: parts to be omitted

Figure 4-5: Definition of a Filter Using the Filter Kit

In the example given, the user wants to see information about called functions by looking at their code.

for the designer (e.g., which meaningful artifacts can be constructed, how and which blocks can be combined with each other).

Design kits are an instance of intelligent support systems (Figure 3-1) and should be integral parts of future computer systems. Each system that allows user modifiability should have an associated design kit. Design kits can contribute to (at least partially) resolving the basic design conflict between generality and power versus ease of use. Design kits provide prototypical solutions and examples which can be modified and extended to achieve a new goal instead of starting from scratch; they support a "copy&edit" methodology for constructing systems through reuse and redesign of existing components [Fischer, Lemke, Rathke 87].

In the following WIDES [Fischer, Lemke 87a; Fischer, Lemke 87b], a design kit for the construction of window systems using WUSP will be described in greater detail. The goals of WIDES are to reduce the knowledge required to use the components, to support the learning process and to provide guidelines to structure a toolkit in such a way that useful work can be done when only a small part of it is known. WIDES provides a safe learning environment in which no fatal errors are possible and in which enough information is provided in each situation to make sure that there is always a way to proceed. The design kit allows users to create simple window types for their applications.

Description of WIDES. The initial state of the system is shown in Figure 4-6. It is a window with four panes:

- a *code* pane that displays the *current* definition of the window type as program text,
- a menu of *suggestions* for enhancements of the window type,
- a *history* list,
- a menu of general *operations*.

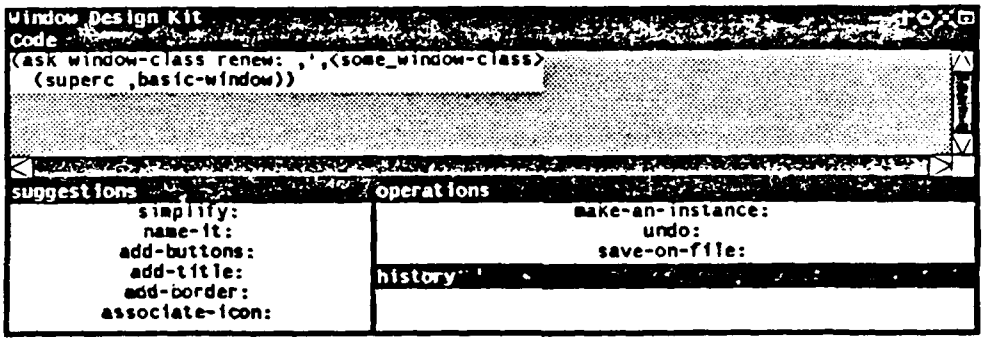

Figure 4-6: Initial State of WIDES

Figure 4-7 shows a window and an icon of the selected type. The history list shows the operations carried out by the user. The contents of the suggestions pane changes dynamically, and it lists only those operations which are meaningful at a specific time in the design task. The code pane shows the generated code.

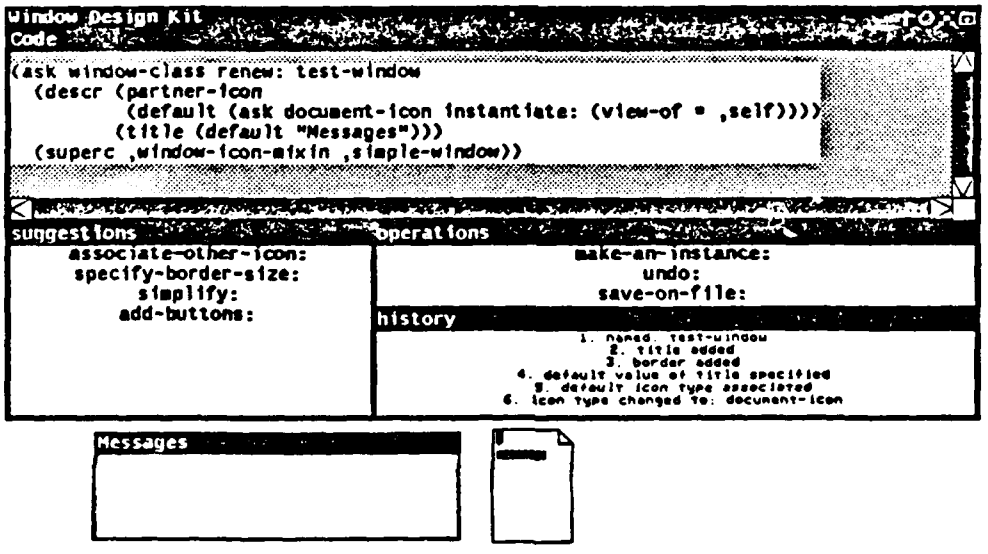

Figure 4-7: The Window and its Associated Icon

A more complex modification is demonstrated in Figure 4-8. Windows can be associated with push buttons such as those in the upper right corner of the window design kit window. Clicking the button with the mouse causes a message to be sent to the window. As an extension of the push buttons in the title bar supplied by default (the two right-most ones), a button for burying the window is to be added. After selecting "**add-more-buttons-to-title-bar:**" from the suggestions menu, the user is asked to choose a button icon and a message from two menus. The bury button appears as the leftmost button in the "**Messages**" window in Figure 4-8. The "**save-on-file:**" operation may be used to save the final definition for later use.

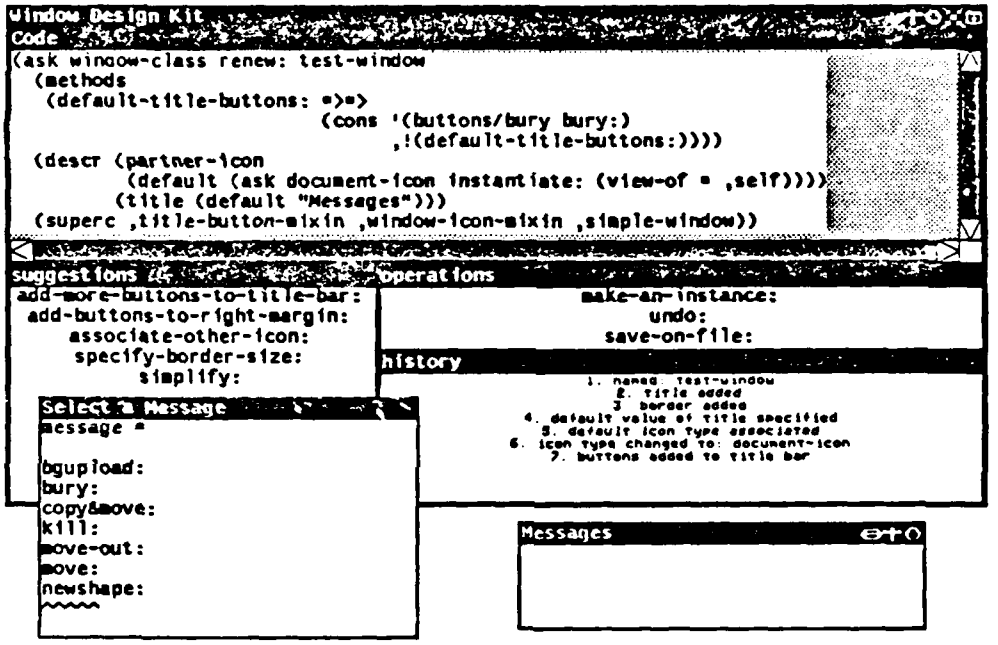

Figure 4-8: Adding a Button to the Title Bar

Although not much code is being generated by the system because it can use many high level building blocks (see the code panes in the various stages of the design process), having WIDES represents a significant advantage for the user. In order to construct a new window type, it is no longer necessary to know what building blocks (e.g., the class "**simple-window**") exist, what their names are, and how they are applied. It is no longer necessary to know that new superclasses have to be added to the "**superc**" description of a class. Also, WIDES determines their correct order. The system knows what types of icons are available, how an icon is associated with a window, etc.

User interface techniques like prompting and menus make it easy to experiment in the domain of window construction. The system makes sure that errors are "impossible." This does not mean that these techniques *make sure* that users always *understand* what they are doing.

Methods like this can quite easily be applied in well structured domains like the present one. There are two problems, however, that need to be addressed. The first one is the understanding problem. Seeing

an option in a menu does not imply that its significance is obvious. What does "associate-icon:" mean? What is the function of a window's icon? Another problem may be the sheer number of options. We did not look into this problem because it does not occur in this relatively small system, but future systems may offer hundreds of choice points. For these design kits, a system of reasonable defaults may provide some help if it is combined with a set of predefined samples that are already rather specific starting points.

5. Concluding Remarks

The development of a conceptual framework for hyperknowledge systems and the instantiation of some aspects with actual system building efforts have shown that there is potential to achieve some of the expectations discussed earlier. The research area opens up a large number of interesting cognitive and pragmatic issues. There is no rhetoric how to produce effective material with a hyperknowledge approach.

A critical question is whether designers are willing to spend the extra effort to provide not only one structure (as in linear text arrangements) but a wealth of information about a meta-structure which allows the system or the user to generate a specific structure on demand later on. It also has to be seen whether users are excited about the opportunity to generate a structure by themselves or whether they regard this as an additional demand on their cognitive resources which they would rather not have to do themselves.

There will always be a need for complex systems. The challenge for innovative system designs is to hide complexity from the user while retaining a rich function set. Hyperknowledge architectures promise unique and largely unexplored opportunities to break the "conversation law of complexity" and to create systems which are useful and usable at the same time.

Acknowledgements

I would like to thank my colleagues and students at the University of Colorado, Boulder, especially Andreas Lemke (who developed WIDES and contributed to the development of the program documentation system), Helga Nieper (who developed HELGON), Olivier Brousse and Barbara League (who developed the FLAVARGON system), Emilie Young (who developed the "active examples"), Christian Rathke (who developed OBJTALK and the OBJTALK-BROWSER), and Jim Sullivan, who developed the OBJTALK-NAVIGATOR. Without their contributions, the described research effort would not have been possible.

I would also like to thank all participants of a workshop (jointly organized by W. Kintsch and myself in January 1987; see [Fischer, Nieper 87]) which contributed many interesting thoughts to the topic discussed in this paper. My special thanks go to Stephen Weyer who for many years has pointed out to me interesting questions in relation to hypertext systems and whose research about dynamic books has influenced my own thinking.

The research was supported by: grant No. DCR-8420944 from the National Science Foundation and grant No. MDA903-86-C0143 from the Army Research Institute.

References

[Boecker, Fischer, Nieper 86]
H.-D. Boecker, G. Fischer, H. Nieper, *The Enhancement of Understanding through Visual Representations*, Human Factors in Computing Systems, CHI'86 Conference Proceedings (Boston, MA), ACM, New York, April 1986, pp. 44-50.

[Bush 45]
V. Bush, *As we may Think*, Atlantic Monthly, Vol. 176, No. 7, July 1945, pp. 101-108.

[Cypher 87]
A. Cypher, *Thought-Dumping*, in G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987, ch. 20.

[Delisle, Schwartz 86]
N. Delisle, M. Schwartz, *Neptune: A Hypertext System for CAD Applications*, ACM SIGMOD'86 Proceedings, May 1986, pp. 132-143.

[Ehrlich, Walker 87]
K. Ehrlich, J.H. Walker, *High Functionality, Information Retrieval, and the Document Examiner*, in G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987, ch. 5.

[Fischer 87a]
G. Fischer, *Reuse and Redesign -- A Cognitive View*, IEEE Software, Special Issue on Reusability, July 1987.

[Fischer 87b]
G. Fischer, *A Critic for LISP*, Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milano, Italy), August 1987.

[Fischer, Lemke 87a]
G. Fischer, A.C. Lemke, *Constrained Design Processes: Steps Towards Convivial Computing*, in R. Guindon (ed.), *Cognitive Science and its Application for Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Fischer, Lemke 87b]
G. Fischer, A.C. Lemke, *Construction and Design Kits: Steps Toward Human Problem-Domain Communication*, Paper Submitted to the Journal 'Human-Computer Interaction', Department of Computer Science, University of Colorado, Boulder, CO, 1987.

[Fischer, Lemke, Rathke 87]
G. Fischer, A.C. Lemke, C. Rathke, *From Design to Redesign*, Proceedings of the 9th International Conference on Software Engineering, Computer Society Press of the IEEE, Washington, D.C., March 1987, pp. 369-376.

[Fischer, Lemke, Schwab 85]
G. Fischer, A.C. Lemke, T. Schwab, *Knowledge-Based Help Systems*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 161-167.

[Fischer, Nieper 87]
G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report, No. 87-9, 1987.

[Fischer, Rathke 87]
G. Fischer, C. Rathke, *Beyond Spreadsheets*, Paper Submitted to CHI'88, Department of Computer Science, University of Colorado, Boulder, CO, 1987.

[Fischer, Schneider 84]
G. Fischer, M. Schneider, *Knowledge-Based Communication Processes in Software Engineering*, Proceedings of the 7th International Conference on Software Engineering (Orlando, FA), March 1984, pp. 358-368.

[Furnas 86]
G.W. Furnas, *Generalized Fisheye Views*, Human Factors in Computing Systems, CHI'86 Conference Proceedings (Boston, MA), ACM, New York, April 1986, pp. 16-23.

[Goldberg 84]
A. Goldberg, *Smalltalk-80, The Interactive Programming Environment*, Addison-Wesley Publishing Company, Reading, MA, 1984.

[Halasz, Moran, Trigg 87]
F.G. Halasz, T.P. Moran, R.H. Trigg, *NoteCards in a Nutshell*, Human Factors in Computing Systems and Graphis Interface, CHI+GI'87 Conference Proceedings (Toronto, Canada), ACM, New York, April 1987, pp. 45-52.

[Kay 77] A. Kay, *Microelectronics and the Personal Computer*, Scientific American, 1977, pp. 231-244.

[Kintsch, Mannes 87]
W. Kintsch, S.M. Mannes, *Generating Scripts from Memory*, in J. Hoffmann, E. van der Meer (eds.), *Festschrift for F. Klix*, North-Holland, Amsterdam, 1987, also published as Technical Report No. 87-3, Institute of Cognitive Science, University of Colorado, Boulder, CO.

[Knuth 83]
D.E. Knuth, *Literate Programming*, Technical Report STAN-CS-82-981, Department of Computer Science, Stanford University, September 1983.

[Lewis 87]
C.H. Lewis, *NoPumpG, EXPL, and Spatial Thought Dumper*, in G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987, ch. 13.

[Nieper 87]
H. Nieper, *Information Retrieval by Reformulation: From ARGON to HELGON*, in G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987, ch. 19.

[Patel-Schneider, Brachman, Levesque 84]
P.F. Patel-Schneider, R.J. Brachman, H.J. Levesque, *ARGON: Knowledge Representation Meets Information Retrieval*, Fairchild Technical Report 654, Schlumberger Palo Alto Research, September 1984.

[Rathke 86]
C. Rathke, *ObjTalk: Repraesentation von Wissen in einer objektorientierten Sprache*, PhD Dissertation, Universitaet Stuttgart, Fakultaet fuer Mathematik und Informatik, 1986.

[Simon 81]
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Smolensky et al. 87]
P. Smolensky, B. Fox, R. King, C.H. Lewis, *Computer-Aided Reasoned Discourse, or, How to Argue with a Computer*, Technical Report CU-CS-358-87, Departments of Computer Science and Linguistics, University of Colorado, Boulder, CO, February 1987.

[Stefik et al. 83]
M.J. Stefik, D.G. Bobrow, S. Mittal, L. Conway, *Knowledge Programming in LOOPS: Report on an Experimental Course*, AI Magazine, Fall 1983.

[Trigg, Suchman, Halasz 86]
R.H. Trigg, L.A. Suchman, F.G. Halasz, *Supporting Collaboration in NoteCards*, Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86), MCC, Austin, TX, December 1986, pp. 153-162.

[Weyer 82]
S.A. Weyer, *The Design of a Dynamic Book for Information Search*, International Journal of Man Machine Studies, Vol. 17, No. 1, July 1982, pp. 87-107.

[Weyer 87]
S.A. Weyer, *As We May Learn*, Multimedia in Education: Interfaces to Knowledge, Education Advisory Council Conference Proceedings, Apple Computer, April 1987.

[Weyer, Borning 85]
S.A. Weyer, A.H. Borning, *A Prototype Electronic Encyclopedia*, ACM Transactions on Office Information Systems, Vol. 3, No. 1, January 1985, pp. 63-88.

[Williams 84]
M.D. Williams, *What Makes RABBIT Run?*, International Journal of Man-Machine Studies, Vol. 21, 1984, pp. 333-352.

[Young 87]
E.A. Young, *Using the Full Power of Computers to Learn the Full Power of Computers*, in G. Fischer, H. Nieper (eds.), *Personalized Intelligent Information Systems, Report on a Workshop (Breckenridge, CO)*, Institute of Cognitive Science, University of Colorado, Boulder, CO, Technical Report No. 87-9, 1987, ch. 6.